Simulación de campo eléctrico y líneas equipotenciales en Python

Francisco Carvajal Villegas[1]

[1]*Facultad de Ciencias Físico Matemáticas, Benemérita Universidad Autónoma de Puebla, 72570 Puebla, Pue., Mexico*

13 de agosto de 2025

Resumen: Se creó un programa de computadora utilizando el lenguaje de programación Python, con el que se puede visualizar las líneas equipotenciales de una carga puntual y el campo eléctrico de un sistema de *n* cargas posicionadas de forma arbitraria sobre un plano 2D.

Palabras clave: Python, campo eléctrico, líneas equipotenciales.

Abstract: A computer program was developed using the Python programming language to visualize the equipotential lines of a point charge and the electric field of a system of *n* charges arbitrarily positioned on a 2D plane.

Keywords: Python, electric field, equipotential lines.

1. Introducción

La representación visual del campo eléctrico y las líneas equipotenciales constituye una herramienta esencial para la comprensión de los fenómenos electromagnéticos. Estos conceptos permiten analizar cómo se distribuyen y comportan las fuerzas eléctricas en el espacio debido a la presencia de cargas, así como entender la relación entre el campo eléctrico y el potencial eléctrico. En el ámbito educativo, estas visualizaciones facilitan la interpretación de conceptos abstractos y fortalecen la intuición física de los estudiantes.

Tradicionalmente, estas representaciones se han realizado mediante esquemas teóricos o modelos físicos, los cuales pueden resultar limitados en términos de precisión, flexibilidad o interactividad, especialmente en configuraciones con múltiples cargas. En este contexto, el uso de herramientas computacionales se presenta como una alternativa eficaz para superar estas limitaciones.

Gracias a su sintaxis accesible y su robusto ecosistema de bibliotecas científicas, Python se ha consolidado como un lenguaje idóneo para el desarrollo de simulaciones físicas. En particular, sus capacidades de cálculo numérico y visualización permiten construir modelos que reproducen con fidelidad el comportamiento del campo eléctrico y las superficies equipotenciales asociadas a distribuciones arbitrarias de carga.

El objetivo principal de este trabajo es desarrollar un programa en Python capaz de simular y visualizar tanto el campo eléctrico como las líneas equipotenciales generadas por un sistema de *n* cargas puntuales ubicadas de forma arbitraria sobre un plano bidimensional.

El campo eléctrico es un campo físico que se representa por medio de un modelo que describe la interacción entre cuerpos y sistemas eléctricos. Se puede describir como un campo vectorial en el cual una carga eléctrica puntual de valor q sufre los efectos de una fuerza eléctrica F dada por la siguiente ecuación:

$$\vec{F} = q\vec{E} \tag{1}$$

donde:

$$\vec{E} = \kappa \frac{q\vec{R}}{R^3} \tag{2}$$

con:

$$\kappa = \frac{1}{4\pi\varepsilon_0} \tag{3}$$

en la cual $\varepsilon_0 = 8{,}85 \times 10^{-12} F/m$ es la permitividad del vacío, y $\vec{R}$ el vector de posición de la carga.

La ecuación 1 viene a ser la definición del campo eléctrico y puede interpretarse como una cantidad tal que, cuando se multiplica por una carga puntual, el resultado es la fuerza sobre la carga puntual [4]. $\vec{E}$ se mide en newtons/coulombs. La ecuación (2) constituye una fórmula para calcular $\vec{E}$ en una posición $\vec{r}$ para una carga puntual. El campo eléctrico, al ser una cantidad vectorial, cumple con el principio de superposición, por lo que el campo total eléctrico total de una distribución de cargas puntuales es la suma de la contribución de cada una de ellas:

$$\vec{E} = \kappa \sum_{i=1}^{N} \frac{q\vec{R}_i}{R_i^3} \tag{4}$$

El potencial eléctrico o también trabajo eléctrico en un punto, es el trabajo a realizar por unidad de carga para mover una carga dentro de un campo electrostático desde el punto de referencia

hasta el punto considerado. Dicho de otra forma, es el trabajo que debe realizar una fuerza externa para traer una carga positiva unitaria que desde el punto de referencia hasta el punto considerado, en contra de la fuerza eléctrica y a velocidad constante. Podemos expresarla de la siguiente forma:

$$\phi = \kappa \frac{q}{R} \tag{5}$$

La unidad de medición de potencial escalar recibe el nombre de volt, donde 1 volt = 1 joule/coulomb.

También es necesario dar las siguientes definiciones para vectores.

Sea $\vec{R}$ un vector en dos dimensiones con componentes R_x y R_y en los ejes x y y respectivamente, la magnitud del vector está dada por:

$$R = \sqrt{(R_x)^2 + (R_y)^2} \tag{6}$$

Si queremos conocer el vector unitario de $\vec{R}$ (vector de magnitud 1 en dirección $\vec{R}$), podemos encontrarlo de la siguiente forma.

$$\hat{R} = \frac{\vec{R}}{R} \tag{7}$$

2. Metodología

Se eligió como lenguaje de programación **Python** y fueron utilizadas las siguientes librerías:

- Numpy (Computo y manejo de datos)
- SciPy (Constantes físicas)
- Matplotlib (Gráficas)

Para la producción del código, se utilizó el IDE Spyder.

2.1. Primer caso: Carga puntual

Para empezar, se decidió resolver el caso de una carga puntual situada en el origen.

Comenzamos configurando la constante de Coulomb usando (3) y SciPy [1] para el valor de ε_0:

```
k = 1/(4*np.pi*sc.epsilon_0)
q = 1
x_0, y_0 = 0, 0
```

Después, se construyó la cuadrícula donde se trataron los datos. Esta cuadrícula es conceptualmente un plano cartesiano, donde cada eje se construye con con la función `np.linspace` y luego se crea la matriz de coordenadas con `np,meshgrid`:

```
R = 5
n = 30

x = np.linspace(-R, R, n)
y = np.linspace(-R, R, n)
X, Y = np.meshgrid(x,y)
```

Con se consigue un plano cuadrado de $2R \times 2R$, con un n divisiones.

Luego, se calculó el vector de posición relativa de la carga puntual, que en este caso se calculó de forma general aunque fue convenido que estaría situada en el origen.

Se cuantificaron las componentes del vector y su magnitud (6). También, fue contemplado el error de división entre cero (cuando $r = 0$), modificando este valor por 1×10^{-20}, si es que existe:

```
r_x = X - x_0
r_y = Y - y_0

r = np.sqrt(r_x**2 + r_y**2)
r[r == 0] = 1e-20
```

En el siguiente trozo de código, se computa el valor de cada componente de los vectores de campo eléctrico según (2), sus magnitudes y vectores unitarios (7), además del potencial eléctrico (5):

```
E_x = k * q * (r_x / r**3)
E_y = k * q * (r_y / r**3)
E = np.sqrt(E_x**2 + E_y**2)
E_xu = E_x / E
E_yu = E_y / E

V = k * (q / r)
```

Por último, se visualizan los resultados obtenidos en la misma gráfica. Se optó por usar `plt.quiver` [3] ya que permite esbozar campos vectoriales en 2D representando a los vectores como flechas:

```
plt.figure(figsize=(8, 6))

plt.contour(X, Y, V, levels=20, colors='blue
    ↪ ', linestyles='solid')

plt.quiver(X, Y, E_xu, E_yu, E, cmap="
    ↪ viridis", scale=20)
plt.plot(x_0, y_0, 'bo', label='Carga␣
    ↪ puntual')
plt.title('Instensidad␣de␣campo␣electrico␣de
    ↪ ␣una␣carga␣puntual␣y␣lineas␣
    ↪ equipotenciales')

plt.grid(True)
plt.legend()
plt.show()
```

2.2. Segundo caso: sistema de n cargas en posiciones arbitrarias

Se utilizó la idea del código anterior, ya que como fue establecido antes, por lo que solo se comentarán los cambios importantes. El campo eléctrico cumple con el principio de superposición

(4), debido a esto, se puede aplicar lo previo a cada una de las cargas y después sumar sus contribuciones.

Como se realizó para un caso general, se optó por crear una función que necesitara un argumento `cargas` en forma de lista que contuviera *n* tuplas, correspondientes a la información de cada carga puntual, como ejemplos:

```
q1 = [(-1e-9,0,0)]
q2 = [(-1e-9,-1,0),(-1e-9,1,0)]
```

Donde cada tupla contiene el valor de la carga, la posición inicial en *x*, y la posición inicial en *y* respectivamente.

En este caso, en lugar de calcular directamente el campo eléctrico y el potencial en el `ndarray`, se creó una matriz de ceros del mismo tipo que `X` con `np.zeros_like` [2], que es el eje de las abscisas del plano. Esto es conveniente ya que en vez de calcular el campo para una carga puntual, se debe de hacer para *n* cargas, esto facilita la tarea de sumar las contribuciones de cada una de las cargas al momento de iterar sobre ellas:

```
V = np.zeros_like(X)
Ex = np.zeros_like(X)
Ey = np.zeros_like(X)q1 = [(-1e-9,0,0)]
q2 = [(-1e-9,-1,0),(-1e-9,1,0)]
```

Por principio de superposición, el cómputo de los valores de campo es esencialmente el mismo, solo que se hace para cada una de las cargas y se suma a su matriz correspondiente:

```
for qi, xi, yi in cargas:
    rx = X - xi
    ry = Y - yi

    r = np.sqrt(rx**2 + ry**2)
    r[r == 0] = 1e-20

    Ex += c * qi * (rx / r**3)
    Ey += c * qi * (ry / r**3)
```

En el código anterior, las líneas equipotenciales mas cercanas a la carga se amontonaban, por lo que es difícil apreciar las distintas líneas en esa zona y las flechas de campo eléctrico. Para solucionar esto, se creó una parte del código que elimina las líneas mas cercanas a las cargas, además de calcular las líneas equipotenciales a distancias radiales iguales con respecto a la carga, controlando también el número de líneas visibles en el gráfico:

```
r_min = 0.5
r_max = R
num_niveles = 10
radios = np.linspace(r_min, r_max,
    ↪ num_niveles)

niveles = []
for q in [q for q, _, _ in cargas]:
    niveles_q = c * q / radios
    niveles.extend(niveles_q)
```

Para finalizar, se grafican cada una de las cargas puntuales como puntos en el plano. Color rojo para las cargas positivas y azul para las negativas:

```
    for qi, xi, yi in cargas:
        color = "ro" if qi > 0 else "bo"
        plt.plot(xi, yi, color, label= f'q␣=␣{qi
            ↪ :1e}␣C')
```

Debido a un fallo lógico en el cómputo del potencial para sistemas de dos o mas cargas, actualmente solo se pueden visualizar las líneas equipotenciales de una carga puntual, aunque todavía se cuenta con el valor del potencial en cada punto. Por cuestiones de estilo, se agregan las siguientes líneas de código, para mostrar el título correcto y las líneas, dependiendo del caso:

```
    if len(cargas) == 1:
        plt.contour(X, Y, V, levels = niveles,
            ↪ colors = "blue", linestyles = "
            ↪ solid")
        plt.title('Campo␣electrico␣y␣lineas␣
            ↪ equipotenciales')
    else:
        plt.title('Campo␣electrico')
```

3. Resultados

Se produjo el siguiente código, donde se incluye al inicio `q1, q2, q3, q4` como configuraciones de prueba:

```
#@author: franciscocarvajal
import numpy as np
import scipy.constants as sc
import matplotlib.pyplot as plt

q1 = [(-1e-9,0,0)]
q2 = [(-1e-9,-1,0),(-1e-9,1,0)]
q3 = [(-1e-9,0,2),(-1e-9,-1,0), (-1e-9, 1,0)]
q4 = [(-1e-9,-1,-1),(-1e-9,-1,1), (-1e-9, 1,1),
    ↪ (-1e-9, 1, -1)]

def EV(cargas, R = 5,n = 30, scale = 20):

    k = 1/(4*np.pi*sc.epsilon_0)

    x = np.linspace(-R, R, n)
    y = np.linspace(-R, R, n)
    X, Y = np.meshgrid(x,y)

    V = np.zeros_like(X)
    Ex = np.zeros_like(X)
    Ey = np.zeros_like(X)

    for qi, xi, yi in cargas:
        rx = X - xi
        ry = Y - yi
```

```
    r = np.sqrt(rx**2 + ry**2)
    r[r == 0] = 1e-20

    Ex += k * qi * (rx / r**3)
    Ey += kc * qi * (ry / r**3)

    V = k * (qi / r)

E = np.sqrt(Ex**2 + Ey**2)
Exu = Ex / E
Eyu = Ey / E

r_min = 0.5
r_max = R
num_niveles = 10
radios = np.linspace(r_min, r_max,
    ↪ num_niveles)

niveles = []
for q in [q for q, _, _ in cargas]:
    niveles_q = k * q / radios
    niveles.extend(niveles_q) s
    s

plt.figure(figsize=(8,6))

if len(cargas) == 1:
    plt.contour(X, Y, V, levels = niveles,
        ↪ colors = "blue", linestyles = "
        ↪ solid")
    plt.title("Campo␣electrico␣y␣lineas␣
        ↪ equipotenciales")
else:
    plt.title("Campo␣eleectrico")

plt.quiver(X, Y, Exu, Eyu, E, cmap = "
    ↪ viridis", scale = scale)

for qi, xi, yi in cargas:
    color = "ro" if qi > 0 else "bo"
    plt.plot(xi, yi, color, label= f'q␣=␣{qi
        ↪ :1e}␣C')

plt.grid(True)
#plt.axis("equal")
plt.show()
```

Este código da como resultado las figuras 1, 2, 3 y 4 para las distribuciones de una, dos, tres y cuatro cargas respectivamente.

Las visualizaciones son solo cualitativas, ya que por la construcción del gráfico, no dependen de la escala del plano, por lo que para valores para diferentes escalas a las ya establecidas en el código, se dibujan las mismas representaciones. Sin embargo, los datos de campo eléctrico y potencial eléctrico están guardados en las variables `E` y `V` respectivamente, por lo que se puede trabajar con los datos sin ningún problema.

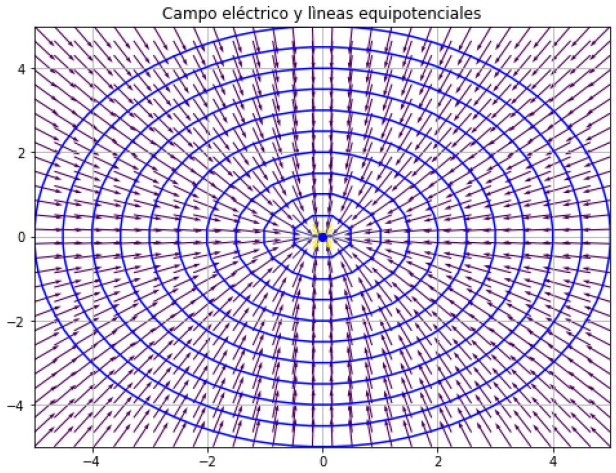

Figura 1: Campo eléctrico y líneas equipotenciales de una carga puntual

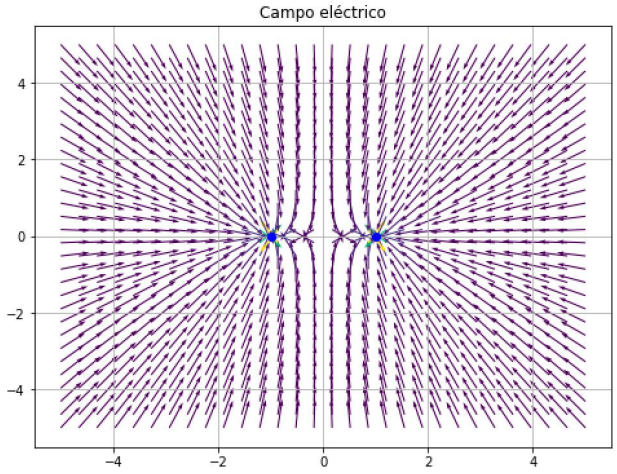

Figura 2: Campo eléctrico de un sistema de dos cargas

Este código y versiones posteriores se podrán encontrar en el siguiente repositorio de github: `https://github.com/FranciscoCarvajal404/Simulacion-de-campo-electrico-y-lineas-equipotenciales`

4. Conclusión

Aunque no se logró la visualización de líneas equipotenciales para distribuciones de n cargas, se puede apreciar la dirección del campo eléctrico en distintos puntos y dichas líneas para el caso de una sola carga. Este programa puede ayudar a los estudiantes en la comprensión de campo eléctrico y potencial de una forma visual, además de proporcionar una solución computacional para su cálculo.

Como continuación, es necesario resolver el problema de la visualización de líneas equipotenciales para sistemas de dos o mas

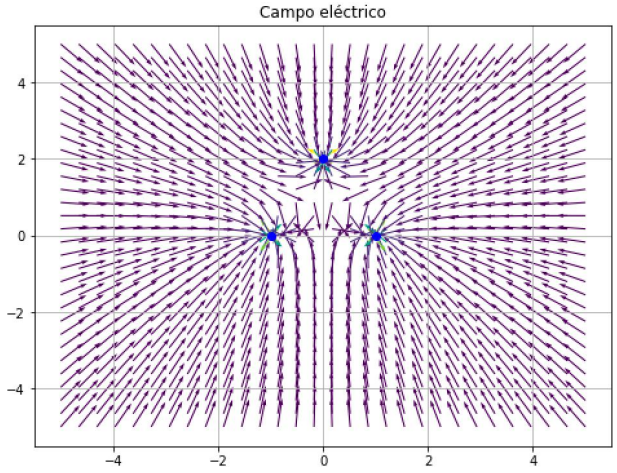

Figura 3: Campo eléctrico de un sistema de tres cargas

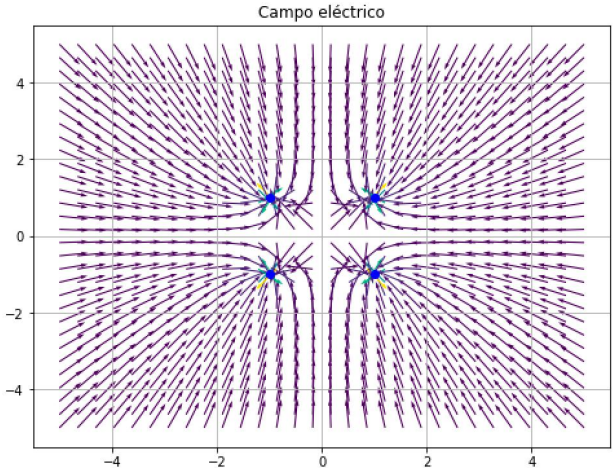

Figura 4: Campo eléctrico de un sistema de cuatro cargas

cargas. Un paso más adelante para la enseñanza de estos temas puede ser el agregar interactividad por medios mas accesibles para el estudiante (como poder arrastrar las cargar con el mouse para ver como cambia el campo en tiempo real, y poder cambiar la magnitud y signo de las cargas solo seleccionándolas en desde una venta), ya que la forma de correr y manipular la simulación es directamente desde el terminal.

Referencias

[1] The SciPy community. *SciPy API reference*. URL: `https://docs.scipy.org/doc/scipy/reference/constants.html`.

[2] NumPy developers. *Numpy API reference*. URL: `https://numpy.org/doc/2.3/user/index.html#user`.

[3] The Matplotlib development team. *Matplotlib API reference*. URL: `https://matplotlib.org/stable/api/index`.

[4] Roald K. Wangsness. *Campos electromagnéticos*. 14.ª ed. Editorial Limusa, 2001.